WLKATA MIROBOT Python SDK使用手册

版本历史

版本号	更新时间	变更记录	编写人
1.0	2021-10-09	文档初始化	邢顺凯(阿凯)
1.1	2021-11-17	添加了传送带的适配	邢顺凯(阿凯)
1.2	2022-04-02	修改部分案例跟代码	邢顺凯(阿凯)

目录

Mirobot 坐标系定义

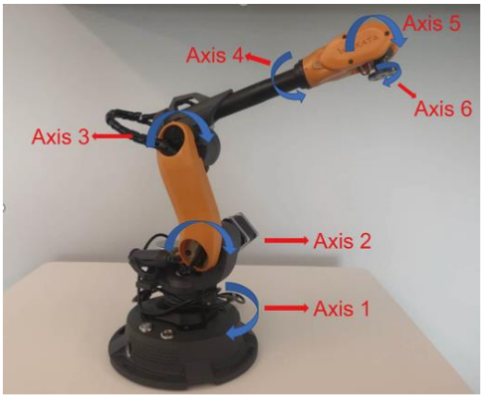


Fig. 1 The six axes of Mirobot

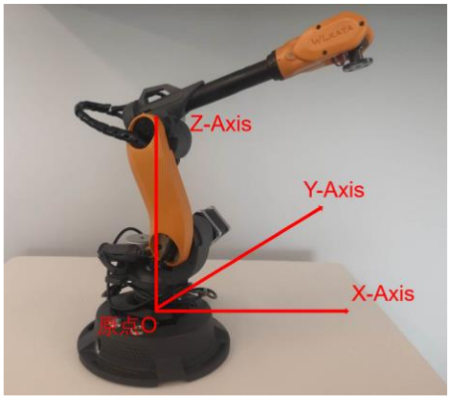


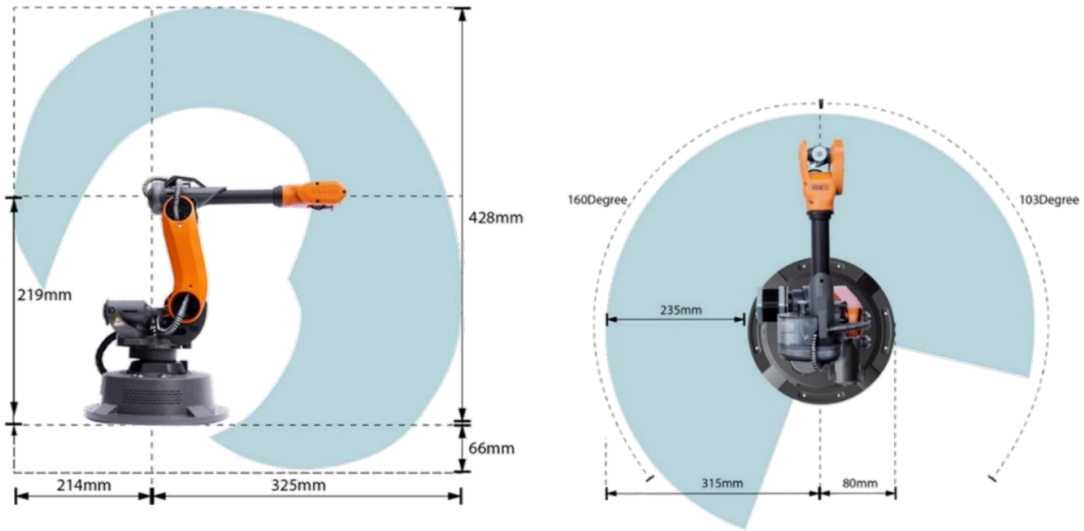
Fig. 2 Robot coordinate system and origin

Mirobot六自由度机械臂关节(轴)定义如左图所示， 关节角度单位为°。 另外滑台/传送带属于第七轴，位置单位为mm。

机械臂的机械臂基坐标系如右图所示。

工具坐标系定义：默认机械臂第 6 关节末端法兰盘底面中心点为工具坐标原点，如果安装其他工具，则可以根据需要选取工具上某一点为参考点，并设定其相对于法兰盘中心的 XYZ 偏移量，则笛卡尔坐标模式下控制的机械臂末端位姿即为该点位姿。

机械臂末端的工作空间:



安装Mirobot Python SDK

Mirobot Python SDK的代码仓库的名字叫做`wlkata-mirobot-python`

[Github代码仓库]

Windows操作系统， 通过pip3安装mirobot python sdk

```
pip install wlkata-mirobot-python
```

Ubuntu操作系统, 通过pip3安装mirobot python sdk

```
sudo pip3 install wlkata-mirobot-python
```

创建机械臂对象

导入依赖

```
from wlkata_mirobot import WlkataMirobot
```

创建机械臂对象

```
arm = WlkataMirobot()
```

默认为自动寻找设备模式，扫描串口设备并尝试通信， 如果得到机械臂状态信息则创建链接。

如果想要手动设定mirobot的设备号, 可以传入端口号`portname`

Windows平台下端口名称格式为 `COM` + `编号`, 示例如下:

```
# - Windows: COM + ID, exp: 'COM3'
arm = WlkataMirobot(portname='COM3')
```

Linux平台下端口名称格式为`/dev/ttyUSB` + `编号`，示例如下:

```
arm = WlkataMirobot(portname='/dev/ttyUSB0')
```

本文后续`arm`均指代mirobot对象.

机械臂归零(Homing)

机械臂上电的时候， 或者刚开始建立串口连接的时候， 机械臂的各个轴处于锁定状态。 必须执行机械臂归零的动作， 才能解锁各轴， 然后才能执行动作。

API-`home` 机械臂同时归零

机械臂本体或者 机械臂本体+滑台同时归零。

函数原型

```
def home(self, has_slider=False):
```

输入参数

- `has_slider`：`bool` 是否有滑轨

使用示例

默认值为`False`，默认只有机械臂本体， 直接执行

```
arm.home()
```

如果是有滑台的情况， 需要将 `has_slider`参数设置为`True`

```
arm.home(has_slider=True)
```

API-`home_1axis` 单轴归零

单轴归零

函数原型

```
def home_1axis(self, axis_id):
```

输入参数

- `axis_id`:`int` 轴的编号

  取值范围 $[1, 7]$

使用示例

```
# 关节1归零
arm.home_1axis(1)
```

API-`home_slider` 滑台归零

滑台单独归零

函数原型

```
def home_slider(self)
```

使用示例

```
arm.home_slider()
```

示例脚本-机械臂归零

```
'''
机械臂回归机械零点与状态查询
'''
from wlkata_mirobot import WlkataMirobot
```

```python
import time

print("实例化Mirobot机械臂实例")
arm = WlkataMirobot()

# 机械臂Home 多轴并行
print("机械臂Homing开始")
# 注:
# - 一般情况, 在无第七轴的时候,  直接执行 arm.home() 即可,
# 参数has_slider默认为False。
# - 如有有滑台(第七轴), 将has_slider设置为True
arm.home()
# arm.home(has_slider=False)
# arm.home(has_slider=True)
print("机械臂Homing结束")

# 状态更新与查询
print("更新机械臂状态")
arm.get_status()
print(f"更新后的状态对象: {arm.status}")
print(f"更新后的状态名称: {arm.status.state}")
```

限位(Limit)

机械臂运动触发软限位后，机械臂停止运动，此时反方向运动即可解除;
机械臂触发硬限位后机械臂锁死，需重启机械臂;

API-`set_hard_limit` 设置硬限位

设置是否开启限位开关的硬限位

函数原型

```python
def set_hard_limit(self, enable):
```

输入参数

- `enable`:`bool` 使能
  - `True`： 开启硬限位
  - `False`:关闭硬限位

使用示例

```python
arm.set_hard_limit(True)
```

API-`set_soft_limit` 设置软限位

设置是否开启软限位

函数原型

```
def set_soft_limit(self, enable)
```

输入参数

- `enable`：`bool` 使能
  - `True`： 开启软限位
  - `False`: 关闭软限位

使用示例

```
arm.set_soft_limit(True)
```

各轴解锁(Unlock All Axis)

API-`unlock_all_axis` 各轴解锁

解除各轴锁定状态， 此API慎用。

函数原型

```
def unlock_all_axis(self)
```

使用示例

```
arm.unlock_all_axis()
```

设置默认运动速度(Set Speed)

API-`set_speed` 设置默认运动速度

函数原型

```
def set_speed(self, speed):
```

输入参数

- `speed`：`float` 移动速度， 单位mm/min

  取值范围$(0, 3000]$，速度默认值为2000

使用示例

```
arm.set_speed(2000)
```

机械臂回零(Go To Zero)

API-`go_to_zero` 机械臂回零

控制机械臂的各轴进入名义上的零点.

函数原型

```
def go_to_zero(self):
```

使用示例

```
arm.go_to_zero()
```

设置关节角度(Set Joint Angle)

设置关节角度

API-`set_joint_angle`

函数原型

```
def set_joint_angle(self, joint_angles, is_relative=False, speed=None,
wait_ok=None):
```

输入参数

- `joint_angles`：`dict` 目标关节角度字典

  key是关节的ID号，取值范围$[1, 7]$

  对于关节1到关节6 value是角度(单位°), 对于第七轴单位则是mm

  举例: `{1:45.0, 2:-30.0}`
- `speed`：`float` 移动速度， 单位mm/min
- `is_relative` ：`bool` 是否为相对移动
- `wait_ok`： `bool` 是否等待机械臂接收到指令后返回“ok”信息。

使用示例

```
target_angles = {1:90.0, 2:30.0, 3:-20.0, 4:10.0, 5:0.0, 6:90.0}
arm.set_joint_angle(target_angles)
```

示例脚本-设置机械臂关节角度

`set_joint_angle.py`

```
'''
设置机械臂关节的角度, 单位°
'''
from wlkata_mirobot import WlkataMirobot
import time
arm = WlkataMirobot()
arm.home()

# 设置单个关节的角度
print("测试设置单个关节的角度")
arm.set_joint_angle({1:100.0})
print("动作执行完毕")
# 状态查询
print(f"状态查询: {arm.get_status()}")
# 停顿2s
time.sleep(2)

# 设置多个关节的角度
print("设置多个关节的角度")
target_angles = {1:90.0, 2:30.0, 3:-20.0, 4:10.0, 5:0.0, 6:90.0}
arm.set_joint_angle(target_angles)
print("动作执行完毕")
# 状态查询
print(f"状态查询: {arm.get_status()}")
# 停顿2s
time.sleep(2)

```

滑台控制(Slider Control)

API-`set_slider_posi` 设置滑台位置

函数原型

```
def set_slider_posi(self, d, speed=None, is_relative=False, wait_ok=True):
```

输入函数

- `d`：`float` 滑台的位置， 单位mm
- `speed`：`float` 移动速度， 单位mm/min
- `is_relative` ：`bool` 是否为相对移动
- `wait_ok`： `bool` 是否等待机械臂接收到指令后返回“ok”信息。

使用示例

```
print('设置滑台的位置 300mm, 速度 2000 mm/min')
arm.set_slider_posi(300, speed=2000)
```

示例脚本-滑台位置控制

`slider.py`

```python
'''
滑台控制示例
'''
import time
from wlkata_mirobot import WlkataMirobot

print("实例化Mirobot机械臂实例")
arm = WlkataMirobot()

# 机械臂本体与滑台是否同时Homing
is_home_simultaneously = True
# 归零
if is_home_simultaneously:
    print("本体跟滑台同时归零")
    arm.home(has_slider=True)
else:
    print("机械臂本体Homing")
    arm.home()
    print("滑台Homing")
    arm.home_slider()

print("延时2s")
time.sleep(2)

print('设置滑台的位置 300mm, 速度 2000 mm/min')
arm.set_slider_posi(300, speed=2000)

print("延时2s")
time.sleep(2)

print('设置滑台的位置 100mm')
arm.set_slider_posi(100)

print('设置滑台的位置 相对移动 +50mm')
arm.set_slider_posi(50, is_relative=True)

# 更新机械臂的状态
arm.get_status()
print(f"当前的滑台的位置 :{arm.slider} mm")

```

传送带控制(Conveyor Control)

API - `set_conveyor_range` 设置传送带范围

函数原型

```python
def set_conveyor_range(self, d_min=-30000, d_max=30000):
```

输入函数

- `d_min`：`float` 传送带位置最小值，单位mm

  最小值不能小于`-30000`
- `d_max`：`float` 传送带位置最大值, 单位mm

  最大值不能大于`30000`

使用示例

```
# 设置传动带的运动范围
arm.set_conveyor_range(-30000, 30000)
```

API-`set_conveyor-posi` 设置传送带位置

函数原型

```
def set_conveyor_posi(self, d, speed=None, is_relative=False, wait_ok=True):
```

输入函数

- `d`：`float` 传送带的位置， 单位mm
- `speed`：`float` 移动速度， 单位mm/min
- `is_relative` ：`bool` 是否为相对移动
- `wait_ok`： `bool` 是否等待机械臂接收到指令后返回“ok”信息。

使用示例

```
print('设置传送带的位置 1000mm')
arm.set_conveyor_posi(1000)
```

示例脚本-传送带控制

`conveyor.py`

```
'''
机械臂传送带示例
'''
import time
from wlkata_mirobot import WlkataMirobot

print("实例化Mirobot机械臂实例")
arm = WlkataMirobot()

# 机械臂本体与滑台是否同时Homing
is_home_simultaneously = True

# 注: 机械臂不需要Homing
print("机械臂本体Homing")
```

```python
arm.home()

# 设置传动带的运动范围
arm.set_conveyor_range(-30000, 30000)
print("延时2s")
time.sleep(2)

print('设置传送带的位置 1000mm')
arm.set_conveyor_posi(1000)

print("延时2s")
time.sleep(2)

print('设置传送带的位置 -3000mm')
arm.set_conveyor_posi(-3000)

print("延时2s")
time.sleep(2)

print('设置传送带的位置 相对移动 -1000mm')
arm.set_conveyor_posi(-1000, is_relative=True)

# 更新机械臂的状态
arm.get_status()
print(f"当前的传送带的的位置 :{arm.conveyor} mm")

```

工具-选择末端工具

API-`WlkataMirobotTool`末端工具

在库文件的`WlkataMirobotTool`类里面定义了Mirobot机械臂的末端工具类型， 继承了`Enum`类。

```python
class WlkataMirobotTool(Enum):
    NO_TOOL = 0          # 没有工具
    SUCTION_CUP = 1      # 气泵吸头
    GRIPPER = 2          # 舵机爪子
    FLEXIBLE_CLAW = 3    # 三指柔爪
```

在使用的时候需要引入该类

```python
from wlkata_mirobot import WlkataMirobotTool
```

使用示例

```python
# 吸头工具
WlkataMirobotTool.SUCTION_CUP
```

API-`set_tool_type` 选择末端工具

函数原型

```
def set_tool_type(self, tool, wait_ok=True):
```

输入参数

- `tool`：`WlkataMirobotTool` 末端工具对象
- `wait_ok`： `bool` 是否等待机械臂接收到指令后返回“ok”信息。

使用示例

```
# 更换工具-选择为吸头
arm.set_tool_type(WlkataMirobotTool.SUCTION_CUP)
```

示例脚本-设置末端工具类型

`set_tool_type.py`

```
'''
设置末端工具类型
'''
import time
from wlkata_mirobot import WlkataMirobot, WlkataMirobotTool

print("实例化Mirobot机械臂实例")
arm = WlkataMirobot()

# 机械臂Home 多轴并行
print("机械臂Homing开始")
arm.home()
print("机械臂Homing结束")

# 注: 默认工具为无
# 状态更新与查询
print("更新机械臂状态")
arm.get_status()
print(f"机械臂工具位姿: {arm.cartesian}")

# 更换工具-选择为吸头
arm.set_tool_type(WlkataMirobotTool.SUCTION_CUP)

# 状态更新与查询
print("更新机械臂状态")
arm.get_status()
print(f"机械臂工具位姿(气泵): {arm.cartesian}")

```

工具-设置工具偏移量

API-`set_tool_offset`设置工具偏移量

函数原型

```
def set_tool_offset(self, offset_x, offset_y, offset_z, wait_ok=True):
```

输入参数

偏移量的定义是基于腕关节坐标系原点的。

- `offset_x`：`float` X轴方向上的偏移量， 单位mm
- `offset_y`：`float` Y轴方向上的偏移量， 单位mm
- `offset_z`：`float` Z轴方向上的偏移量， 单位mm
- `wait_ok`：`bool` 是否等待机械臂接收到指令后返回“ok”信息。

使用示例

```
# 偏移量定义，单位mm
offset_x = 0
offset_y = 0
offset_z = -20.0
arm.set_tool_offset(offset_x, offset_y, offset_z)
```

示例脚本-设置工具的偏移量

`set_tool_offset.py`

```
'''
手动设置工具坐标系的偏移量，适用于自制末端的情况。
注：如果是标准的末端工具，建议使用API `set_tool_type`。
'''
import time
from wlkata_mirobot import WlkataMirobot, WlkataMirobotTool

print("实例化Mirobot机械臂实例")
arm = WlkataMirobot()

# 机械臂Home 多轴并行
print("机械臂Homing开始")
arm.home()
print("机械臂Homing结束")

# 注：默认工具为无
# 状态更新与查询
print("更新机械臂状态")
arm.get_status()
print(f"机械臂工具位姿：{arm.cartesian}")

# 偏移量定义，单位mm
offset_x = 0
offset_y = 0
offset_z = -20.0
print(f'手动设置工具坐标系的偏移量 ({offset_x}, {offset_y}, {offset_z}) ')
arm.set_tool_offset(offset_x, offset_y, offset_z)
```

```
28
29  # 状态更新与查询
30  print("更新机械臂状态")
31  arm.get_status()
32  print(f"机械臂工具位姿(气泵): {arm.cartesian}")
33
```

工具-气泵

API-`pump_suction` 气泵吸气

气泵吸气。 如果末端接的是柔性爪， 气泵吸气代表柔性爪外张。

函数原型

```
1  def pump_suction(self):
```

使用示例

```
1  arm.pump_suction()
```

API-`pump_blowing` 气泵吹气

气泵吹气。 如果末端接的是柔性爪， 气泵吸气代表柔性爪内缩。

函数原型

```
1  def pump_blowing(self):
```

使用示例

```
1  arm.pump_blowing()
```

API-`pump_off` 气泵关闭

气泵关闭,。 如果末端接的是柔性爪， 气泵吸气代表柔性爪放松。

函数原型

```
1  def pump_off(self):
```

使用示例

```
1  arm.pump_off()
```

示例脚本-气泵控制

`air_pump.py`

```
'''
气泵控制
'''
from wlkata_mirobot import WlkataMirobot
import time

arm = WlkataMirobot()
arm.home()

# 气泵开启-吸气
arm.pump_suction()
# 等待5s
time.sleep(5)

# 气泵关闭
arm.pump_off()
# 等待5s
time.sleep(2)

# 气泵开启-吹气
arm.pump_blowing()
# 等待5s
time.sleep(5)

# 气泵关闭
arm.pump_off()
# 等待5s
time.sleep(2)
```

工具-舵机夹爪

API-`gripper_open` 夹爪打开

夹爪打开， 此时夹爪间距为最大。

函数原型

```
def gripper_open(self):
```

使用示例

```
arm.gripper_open()
```

API-`gripper_close` 夹爪闭合

夹爪闭合， 此时夹爪间距为最小。

函数原型

```
def gripper_close(self):
```

使用示例

```
arm.gripper_close()
```

API-`set_gripper_spacing` 设置夹爪间距

需要注意的是，夹爪并没有自适应抓取的功能， 因此如果直接调用`gripper_close`去抓取特定的物体会烧坏舵机。

因此更合理的是使用*设置夹爪间距*的API函数， 根据要抓取的目标自动调整夹爪间距。

函数原型

```
def set_gripper_spacing(self, spacing_mm):
```

输入参数

- `spacing_mm`：`float` 爪子间距， 单位mm

使用示例

```
# 设置爪子的间距
spacing_mm = 20.0
arm.set_gripper_spacing(spacing_mm)
```

示例脚本-机械臂爪相关API测试

`gripper.py`

```
'''
机械臂爪相关API测试
'''
from wlkata_mirobot import WlkataMirobot
import time

arm = WlkataMirobot()
arm.home()

# 设置爪子的间距
spacing_mm = 20.0
arm.set_gripper_spacing(spacing_mm)
time.sleep(2)
# 爪子张开
```

```
arm.gripper_open()
time.sleep(2)
# 爪子闭合
arm.gripper_close()
time.sleep(2)
```

机械臂状态信息

API-`get_status`-获取并更新机械臂状态

更新当前机械臂的状态。在访问机械臂当前状态相关的属性之前， 记得要执行该函数。

函数原型

```
def get_status(self, disable_debug=False):
```

使用示例

```
arm.get_status()
```

机械臂状态符

获取当前机械臂状态的字符串描述。

```
arm.status.state
```

- `“Alarm”`：报警状态， 自锁状态
- `“Home”`： 归零中
- `"Idle"`：机械臂空闲
- `"Busy"`：机械臂运动中

关节角度

获取机械臂各个关节的角度

```
# 关节1的角度, 单位°
arm.angle.joint1
# 关节2的角度, 单位°
arm.angle.joint2
# 关节3的角度, 单位°
arm.angle.joint3
# 关节4的角度, 单位°
arm.angle.joint4
# 关节5的角度, 单位°
arm.angle.joint5
# 关节6的角度, 单位°
arm.angle.joint6
# 滑台的位置， 单位mm
```

```
arm.slider
```

工具位姿

获取工具在机械臂基坐标系下的位姿(6DoF)

```
# 工具在机械臂坐标系下的X轴坐标
arm.pose.x
# 工具在机械臂坐标系下的Y轴坐标
arm.pose.y
# 工具在机械臂坐标系下的Z轴坐标
arm.pose.z
# 工具的姿态 横滚角, 单位°
arm.pose.roll
# 工具的姿态 俯仰角, 单位°
arm.pose.pitch
# 工具的姿态 偏航角, 单位°
arm.pose.yaw
```

气泵状态

```
# 电磁阀的PWM值
arm.valve_pwm
# 气泵的PWM
arm.pump_pwm
# 机械爪的PWM
arm.gripper_pwm
```

运动模式

```
# 获取机械臂的运动模式
# 布尔值, 代表当前运动模式是相对运动还是绝对运动.
arm.motion_mode
```

示例脚本-获取机械臂状态

`get_status.py`

```
'''
获取机械臂的状态
'''
from wlkata_mirobot import WlkataMirobot
# 创建机械臂对象
arm = WlkataMirobot()
# 机械臂回归零点
arm.home()
# 打印机械臂当前的状态
print("获取机械臂的状态 ?")
print(arm.get_status())
```

工具-末端姿态

API-`set_tool_pose`

设置机械臂工具在机械臂基坐标系下的位姿。

函数原型

```
def set_tool_pose(self, x=None, y=None, z=None, roll=0.0, pitch=0.0, yaw=0.0, \
                  mode='p2p', speed=None, is_relative=False, wait_ok=True):
```

输入参数

- `x`：`float` 工具x坐标
- `y`：`float` 工具y坐标
- `z`：`float` 工具z坐标
- `roll`：`float` 工具横滚角
- `pitch`：`float` 工具俯仰角
- `yaw`： `float` 工具偏航角
- `mode`：`str` 运动模式
  - `'p2p'` 点到点快速运动
  - `'linear'`：直线插补
- `speed`：`float` 移动速度， 单位mm/min
- `is_relative` ：`bool` 是否为相对移动
- `wait_ok`： `bool` 是否等待机械臂接收到指令后返回“ok”信息。

使用示例

设置工具的坐标

```
arm.set_tool_pose(200,  20, 230)
```

设置工具的坐标+欧拉角

```
arm.set_tool_pose(150,  -20,  230, roll=30.0, pitch=0, yaw=45.0)
```

设置工具的坐标(相对运动)

```
arm.set_tool_pose(5,  0, 10, is_relative=True)
```

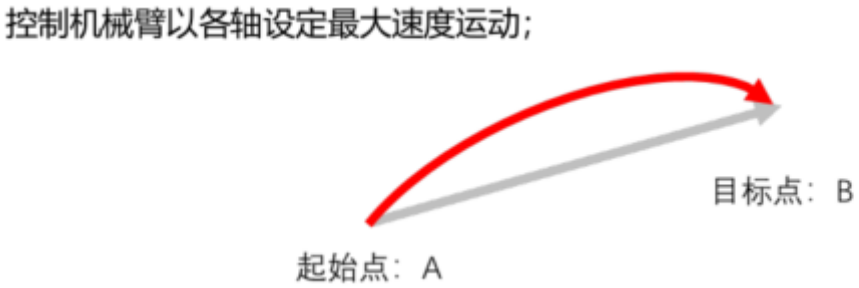
示例脚本-机械臂工具位姿控制

`set_tool_pose.py`

```
'''
机械臂工具位姿控制, 点控 point to point
'''
```

```python
from wlkata_mirobot import WlkataMirobot
import time
# 创建机械臂
arm = WlkataMirobot()
# Homing
arm.home()

print("运动到目标点 A")
arm.set_tool_pose(200,  20, 230)
print(f"当前末端在机械臂坐标系下的位姿 {arm.pose}")
time.sleep(2)


print("运动到目标点 B")
arm.set_tool_pose(200,  20, 150)
print(f"当前末端在机械臂坐标系下的位姿 {arm.pose}")
time.sleep(2)

print("运动到目标点 C, 指定末端的姿态角")
arm.set_tool_pose(150,  -20,  230, roll=30.0, pitch=0, yaw=45.0)
print(f"当前末端在机械臂坐标系下的位姿 {arm.pose}")
time.sleep(2)

print("机械臂回零")
arm.go_to_zero()
```

轨迹规划-点到点(P2P)



API-`p2p_interpolation`

轨迹规划， 快速运动模式。 不在乎中间的轨迹。

函数原型

```python
def p2p_interpolation(self, x=None, y=None, z=None, a=None, b=None, c=None, \
                      speed=None, is_relative=False, wait_ok=None):
```

输入参数

- `x`：`float` 工具x坐标
- `y`：`float` 工具y坐标
- `z`：`float` 工具z坐标
- `a`：`float` 工具横滚角

- `b`：`float` 工具俯仰角
- `c`： `float` 工具偏航角
- `speed`：`float` 移动速度， 单位mm/min
- `is_relative` ：`bool` 是否为相对移动
- `wait_ok`： `bool` 是否等待机械臂接收到指令后返回“ok”信息。

使用示例

```
arm.p2p_interpolation(100,  100, 150)
```

```
x, y, z = 100,  -100, 150
roll, pitch, yaw = 30.0, 0, 45.0
arm.p2p_interpolation(x, y, z, roll, pitch, yaw)
```

示例脚本-点到点快速移动

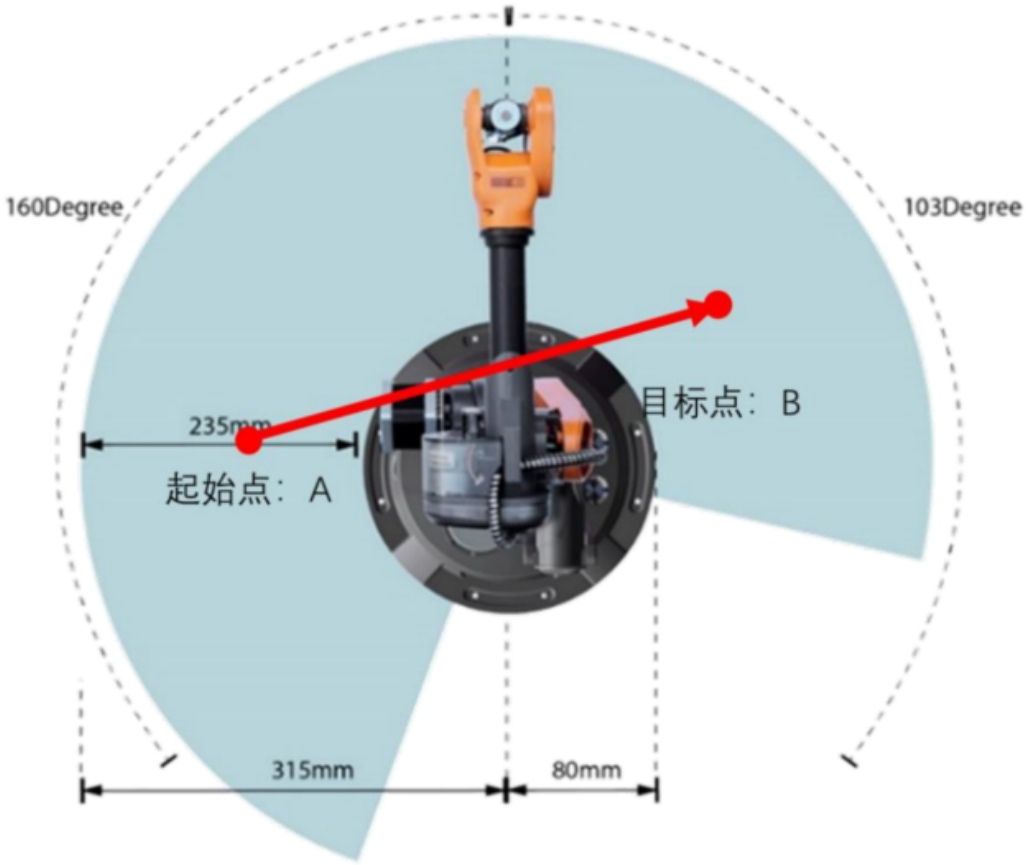
`p2p_interpolation.py`

```
'''
机械臂工具位姿控制
插补算法: 点到点快速移动(p2p point-to-point)
'''
from wlkata_mirobot import WlkataMirobot
import time
# 创建机械臂
arm = WlkataMirobot()
# Homing
arm.home()

print("运动到目标点 A")
arm.p2p_interpolation(100,  100, 150)
print(f"当前末端在机械臂坐标系下的位姿 {arm.pose}")
time.sleep(2)


print("运动到目标点 B")
x, y, z = 100,  -100, 150
roll, pitch, yaw = 30.0, 0, 45.0
arm.p2p_interpolation(x, y, z, roll, pitch, yaw)
print(f"当前末端在机械臂坐标系下的位姿 {arm.pose}")
time.sleep(2)
```

轨迹规划-直线插补

直线插补运动（末端运动轨迹为直线）

使用时需注意机械臂的工作空间为环形，当出现轨迹超出工作空间时，机械臂无法正常执行该指令，即使起始点 A 与目标点 B 都在工作空间内。（如下图所示），此种情况请使用P2P快速运动；



API-`linear_interpolation` 线性插补

函数原型

```
def linear_interpolation(self, x=None, y=None, z=None, a=None, b=None,
c=None, speed=None, is_relative=False, wait_ok=None):
```

输入参数

- `x`：`float` 工具x坐标
- `y`：`float` 工具y坐标
- `z`：`float` 工具z坐标
- `a`：`float` 工具横滚角
- `b`：`float` 工具俯仰角
- `c`：`float` 工具偏航角
- `speed`：`float` 移动速度， 单位mm/min

- `is_relative` ：`bool` 是否为相对移动
- `wait_ok`： `bool` 是否等待机械臂接收到指令后返回“ok”信息。

使用示例

```
arm.linear_interpolation(200,  -50, 150)
```

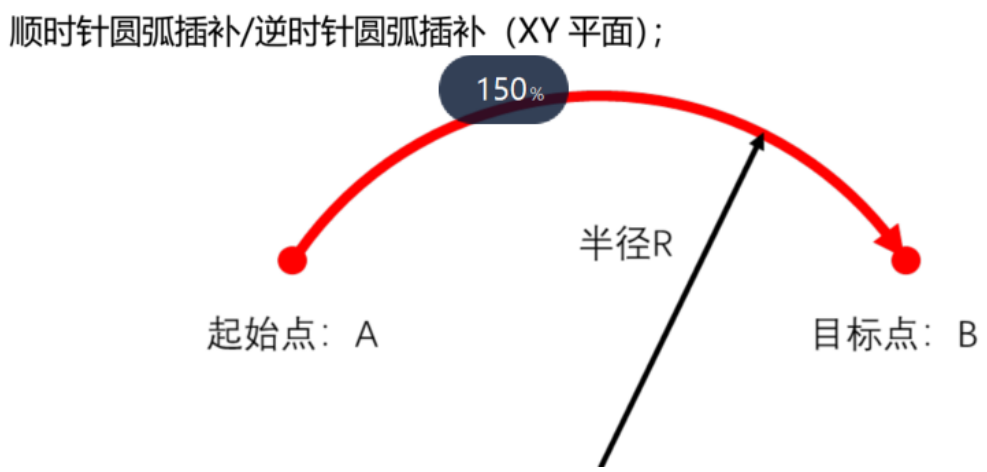
示例脚本-直线插补

`linear_interpolation.py`

```
'''
机械臂工具位姿控制
插补算法: 直线插补(linear_interpolation.)
'''
from wlkata_mirobot import WlkataMirobot
import time
# 创建机械臂
arm = WlkataMirobot()
# Homing
arm.home()

print("运动到目标点 A")
arm.linear_interpolation(200,  50, 150)
print(f"当前末端在机械臂坐标系下的位姿 {arm.pose}")
time.sleep(2)

print("运动到目标点 B")
arm.linear_interpolation(200,  -50, 150)
print(f"当前末端在机械臂坐标系下的位姿 {arm.pose}")
time.sleep(2)

```

轨迹规划-圆弧插补

API-`circular_interpolation` 圆弧插补



在XY平面上, 从当前点运动到相对坐标(ex, ey).半径为radius， 轨迹为一段圆弧， `is_cw`决定圆弧是顺时针还是逆时针.

起点与终点间的距离应小于等于所设定圆弧半径值的 2 倍（圆弧直径）。

函数原型

```
def circular_interpolation(self, ex, ey, radius, is_cw=True, speed=None,
wait_ok=None):
```

输入参数

设置当前末端位置为圆弧起始点。

- `ex` : `float` 圆弧终止点的x坐标(相对位置), 单位mm
- `ey` : `float` 圆弧终止点的y坐标(相对位置), 单位mm
- `radius` : `float` 圆弧半径， 单位mm
- `speed` : `float` 移动速度， 单位mm/min
- `wait_ok` : `bool` 是否等待机械臂接收到指令后返回“ok”信息。

使用示例

```
ex, ey = (0, -80)    # 末端目标坐标, 单位mm(相对于当前点)
radius = 100         # 半径, 单位mm
is_cw = False        # 运动方向 True: 顺时针, False: 逆时针
arm.circular_interpolation(ex, ey, radius, is_cw=is_cw)
```

示例脚本-圆弧插补

`circular_interpolation.py`

```
'''
机械臂工具位姿控制
插补算法: 圆弧插补(circular interpolation.)
'''
from wlkata_mirobot import WlkataMirobot
import time
# 创建机械臂
arm = WlkataMirobot()
# Homing
arm.home()

print("运动到目标点 A")
arm.set_tool_pose(200,  40, 150)
print(f"当前末端在机械臂坐标系下的位姿 {arm.pose}")
time.sleep(2)


print("运动到目标点 B(圆弧插补)")
ex, ey = (0, -80)    # 末端目标坐标, 单位mm(相对于当前点)
radius = 100         # 半径, 单位mm
is_cw = False        # 运动方向 True: 顺时针, False: 逆时针
arm.circular_interpolation(ex, ey, radius, is_cw=is_cw)
print(f"当前末端在机械臂坐标系下的位姿 {arm.pose}")
```

```
time.sleep(2)

```

轨迹规划-门式轨迹规划

门型轨迹运动；

API-`set_door_lift_distance` 设置抬起高度

函数原型

```
def set_door_lift_distance(self, lift_distance):
```

输入参数

- `lift_distance`：`float` 抬起高度， 单位mm

使用示例

```
# 设置门式轨迹规划抬升高度
arm.set_door_lift_distance(50)
```

API-`door_interpolation` 门式轨迹插补

函数原型

```
def door_interpolation(self, x=None, y=None, z=None, a=None, b=None, c=None,
\
                       speed=None, is_relative=False, wait_ok=None):
```

输入参数

- `x`：`float` 工具x坐标
- `y`：`float` 工具y坐标
- `z`：`float` 工具z坐标
- `a`：`float` 工具横滚角
- `b`：`float` 工具俯仰角
- `c`：`float` 工具偏航角

- `speed`：`float` 移动速度， 单位mm/min
- `is_relative` ：`bool` 是否为相对移动
- `wait_ok`： `bool` 是否等待机械臂接收到指令后返回“ok”信息。

使用示例

```
arm.door_interpolation(200, -40, 150)
```

示例脚本-门式插补

`door_interpolation.py`

```
'''
机械臂工具位姿控制
插补算法: 门式插补(door interpolation.)
'''
from wlkata_mirobot import WlkataMirobot
import time
# 创建机械臂
arm = WlkataMirobot()
# Homing
arm.home()
# 设置门式轨迹规划抬升高度
arm.set_door_lift_distance(50)

print("运动到目标点 A")
arm.set_tool_pose(200,  40, 150)
print(f"当前末端在机械臂坐标系下的位姿 {arm.pose}")
time.sleep(2)


print("运动到目标点 B(门型插补)")
arm.door_interpolation(200, -40, 150)
print(f"当前末端在机械臂坐标系下的位姿 {arm.pose}")
time.sleep(2)
```